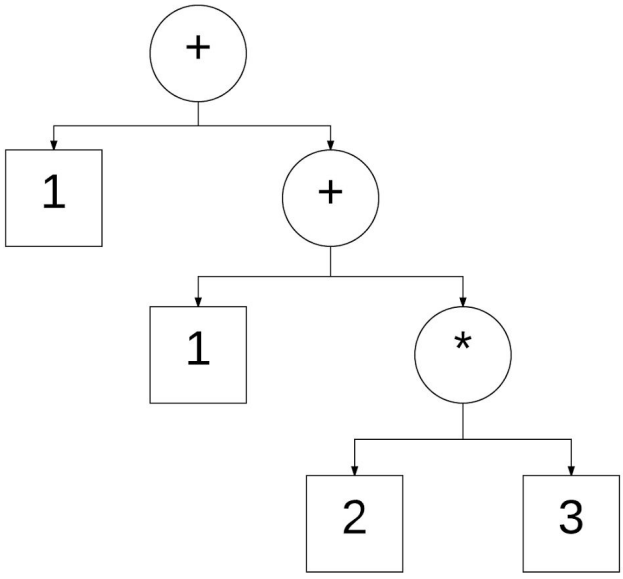
Hunter Dubel
Software Engineer Project: Data Structure
Project Index: 13

For this project, I used the assumption that the equation would only contain the operators for addition and multiplication. The following was designed around and tested on the arithmetic expression of 1+2*3+1.

Scenario:
You want to use a binary tree to encode infix arithmetic expressions on integers. Operations are addition and multiplication. Complete the following tasks.

1. Draw a picture of what the tree looks like.



2. Write a class definition.
   See included code.
3. Write an evaluate() member function.
   See included code.
4. How would you make your evaluate() iterative instead of recursive?
   To make the evaluate() iterative instead of recursive, the amount of operators would need to be counted when evaluate() is first called. Evaluate() would contain a for loop that would run once for each operator found in the equation.

Code

```
# Expressions.py
# Software Engineer Project: Data Structure
# Hunter Dubel

# Draft a response to the following scenario and upload your response.
```

```
# Scenario:

# You want to use a binary tree to encode infix arithmetic expressions on integers.
Operations are addition and multiplication. Complete the following tasks.

# Draw a picture of what the tree looks like.
# Write a class definition.
# Write an evaluate() member function.
# How would you make your evaluate() iterative instead of recursive?


class Expression:
	exArr1 = []
	exArr2 = []
	tmpSum = 0;

	def __init__(self, equation):
		self.equation = equation
		self.exArr1 = list(equation)

	def evaluate(self):
		print("Arr1 " + str(self.exArr1))
		print("Arr2 " + str(self.exArr2) + "\n")
		if '*' in self.exArr1:		#Check if there is multiplication due to Order
of Ops
			print("preArr1 " + str(self.exArr1))
			print("preArr2 " + str(self.exArr2) + "\n")
			self.exArr2.append(self.exArr1.pop(0)) #If there is a *, since it is
infix, we transition the first number over.
			print("postArr1 " + str(self.exArr1))
			print("postArr2 " + str(self.exArr2) + "\n")
			if '*' in self.exArr1[0]:    #If the first symbol is *
				self.exArr1.pop(0) #Remove the symbol. No longer needed.
				num1 = int(self.exArr1.pop(0))
				num2 = int(self.exArr2.pop())
				tmpSum = num1 * num2
				self.exArr1 = self.exArr2 + [tmpSum] + self.exArr1 #Restore the
equation
				self.exArr2 = []
				self.evaluate()
			else:
				self.exArr2.append(self.exArr1.pop(0)) #If there is a +, we
need to remove it
				self.evaluate()
		elif '+' in self.exArr1:
			self.exArr2.append(self.exArr1.pop(0))
			if '+' in self.exArr1[0]:
				self.exArr1.pop(0)
				num1 = int(self.exArr1.pop(0))
				num2 = int(self.exArr2.pop())
				tmpSum = num1 + num2
				self.exArr1 = self.exArr2 + [tmpSum] + self.exArr1 #Restore the
equation
				self.exArr2 = []
				self.evaluate()
			else:
				self.evaluate()

problem1 = Expression('5+2*3+1')
problem1.evaluate()
```